Data Parallel Algorithms for Engineering Applications

Rolf-Dieter Fiebrich
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142

E/G87-1

Abstract

This paper shows how computation-intensive engineering problems can be computed efficiently on a massively parallel computer. Designed with tens of thousands of processing elements, these machines now offer substantially improved computation times and improved cost performance, and in the future promise to rapidly reach higher performance levels. We will discuss the ease of developing data parallel algorithms for VLSI circuit placement, VLSI circuit simulation and seismic migration.

1 Introduction

Massively parallel computers with tens of thousands of processing elements which can operate in parallel are entering the market place. Because of the regularity and simplicity of the design, these machines offer a substantial cost performance improvement over conventional machines. Furthermore, these machines promise to reach substantially higher absolute performance levels in the coming years than conventional supercomputers.

Key questions are often raised: Are computation-intensive applications parallelizable such that these machines can be used efficiently? Can these machines be programmed with reasonable effort?

This paper will address these questions by discussing in some detail three examples of computation intensive algorithms from the engineering domain, namely VLSI circuit placement, VLSI circuit simulation and seismic migration. A goal is to provide some basic insight into the process of designing algorithms for a massively parallel computer from which the reader can extrapolate how other applications can be computed on such a machine.

Virtually all computation-intensive algorithms deal with large amounts of data. VLSI design deals with the placement of thousands of circuit elements or with the simulation of networks of hundreds of thousands of transistors. Seismic migration computes millions of gridpoints of a subsurface image, etc. Massively parallel computing takes advantage of the fact that in those applications, thousands of data objects can be operated on in parallel.

Essential to the ease of parallel programming is a simple model for parallel computation which is efficiently implemented by the underlying hardware. We will use the Connection Machine® system[1] as an example to briefly summarize this model. A programmer starts with familiar concepts for structuring complex data objects such as graphs for representing circuit schematics or arrays for representing images. The new concept is that the variables which represent the vertices of a graph or the elements of an array can be declared to be parallel variables. The other essential concepts are parallel operations which are performed on all parallel variables simultaneously and an operation for selecting a subset of parallel variables in order to restrict parallel operations.

The Connection Machine which supports this model can be viewed as an extension of the memory of a conventional front end, e.g., a VAX, where partitions of the memory have their own execution units which execute parallel operations. In addition, the Connection Machine system has a very sophisticated communications network which permits all parallel execution units to access individual memory locations in the entire machine in parallel.

Parallel variables are stored in individual processing elements, i.e., processor memory pairs. Parallel operations are executed by the parallel execution units. The communication system is involved if compound operations are performed on parallel variables which involve access to other variables; via pointers, for instance.

® Connection Machine is a registered trademark of Thinking Machines Corporation

The remainder of this paper describes the essence of parallel algorithms for three engineering applications, Section 2 describes VLSI Placement, Section 3 describes VLSI Circuit Simulation, and Section 4 deals with seismic migration.

2 VLSI Placement

Semicustom VLSI circuits with more than 10,000 cells or parts have become quite common. Placing this number of parts on a chip such that the area taken up by interconnect wires is minimized is a difficult and time consuming problem.

Many algorithms are in use which tackle this problem. Often these algorithms handle specific placement problems quite well but cannot be easily adjusted to varying conditions due, for instance, to technology changes. A method which seems to yield good optimization results on a broad spectrum of placement problems is the simulated annealing method[2]. Unfortunately simulated annealing is very computation intensive with computation times of 10 to 36 hours on a VAX/780 being reported for circuits with 1500 parts.[3]

Recall that the innerloop of the simulated annealing algorithm consists of generating a new state by moving a part or by swapping parts, calculating the change in cost, and calculating whether to accept or reject the move. The intuitive idea for parallelizing this algorithm is to perform many of these steps in parallel, i.e., move or swap many parts *in parallel*, evaluate the cost change for each of these moves *in parallel* and calculate whether to accept or reject each of the moves *in parallel*. Unfortunately, such parallel moves are in general not independent. For instance if two pairs swap their positions and if those pairs share a common net, then the change in wirelength for one move is generally dependent on whether the other pair actually swaps or not.

The essential data objects used during placement optimization are the netlist of the circuit and the slots on the surface of the chip onto which parts can be placed. We assume a standard cell technology where all parts have identical heights, but various widths. The netlist is represented by assigning each part and each net to a separate processor, that is the data structure which represents a part or a net is stored in the memory of a separate processor. Conventional pointers are used to represent the links between parts and nodes. The chip surface is represented by storing each slot in a separate processing element. Pointers between parts and slots represent the placement of those parts in available slots. Figure 1 shows an example of how a circuit to be placed and a chip surface is mapped onto processing elements.

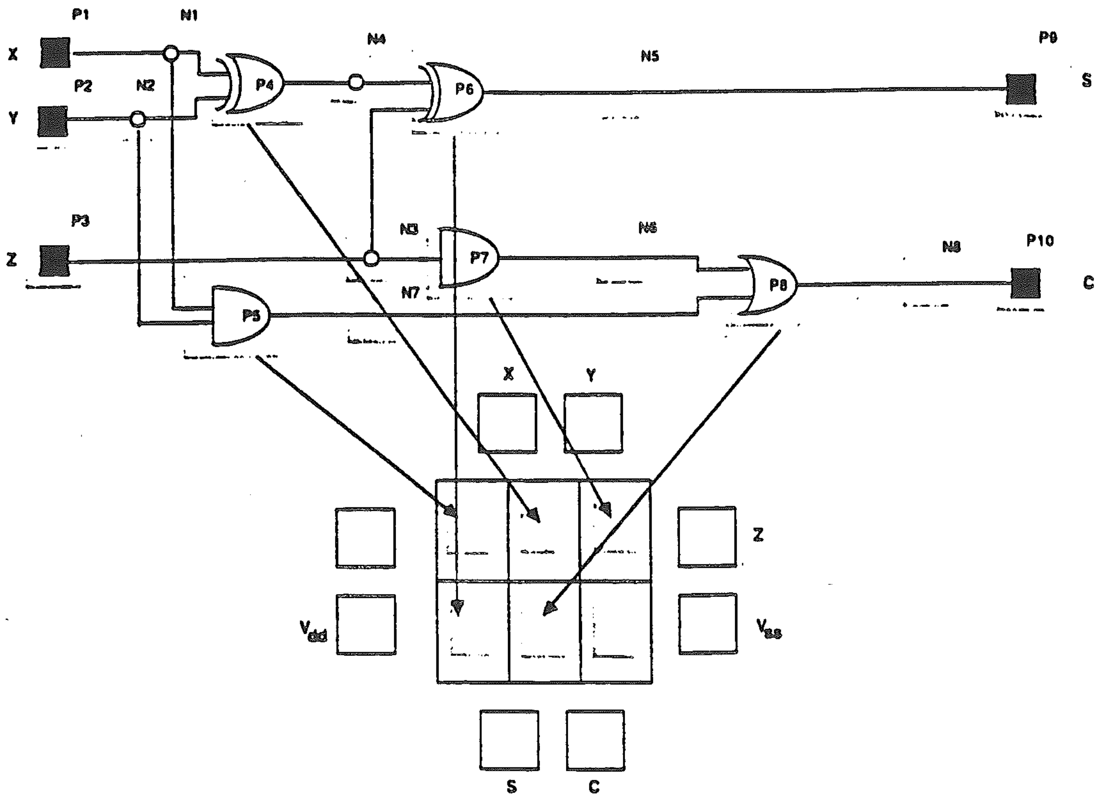

Figure 1: Representation of Circuit and Slots on Surface of Chip on Processing Elements (Shaded areas represent processing elements)

The remainder of this paper describes the essence of parallel algorithms for three engineering applications, Section 2 describes VLSI Placement, Section 3 describes VLSI Circuit Simulation, and Section 4 deals with seismic migration.

2 VLSI Placement

Semicustom VLSI circuits with more than 10,000 cells or parts have become quite common. Placing this number of parts on a chip such that the area taken up by interconnect wires is minimized is a difficult and time consuming problem.

Many algorithms are in use which tackle this problem. Often these algorithms handle specific placement problems quite well but cannot be easily adjusted to varying conditions due, for instance, to technology changes. A method which seems to yield good optimization results on a broad spectrum of placement problems is the simulated annealing method[2]. Unfortunately simulated annealing is very computation intensive with computation times of 10 to 36 hours on a VAX/780 being reported for circuits with 1500 parts.[3]

Recall that the innerloop of the simulated annealing algorithm consists of generating a new state by moving a part or by swapping parts, calculating the change in cost, and calculating whether to accept or reject the move. The intuitive idea for parallelizing this algorithm is to perform many of these steps in parallel, i.e., move or swap many parts *in parallel* , evaluate the cost change for each of these moves *in parallel* and calculate whether to accept or reject each of the moves *in parallel*. Unfortunately, such parallel moves are in general not independent. For instance if two pairs swap their positions and if those pairs share a common net, then the change in wirelength for one move is generally dependent on whether the other pair actually swaps or not.

The essential data objects used during placement optimization are the netlist of the circuit and the slots on the surface of the chip onto which parts can be placed. We assume a standard cell technology where all parts have identical heights, but various widths. The netlist is represented by assigning each part and each net to a separate processor, that is the data structure which represents a part or a net is stored in the memory of a separate processor. Conventional pointers are used to represent the links between parts and nodes. The chip surface is represented by storing each slot in a separate processing element. Pointers between parts and slots represent the placement of those parts in available slots. Figure 1 shows an example of how a circuit to be placed and a chip surface is mapped onto processing elements.

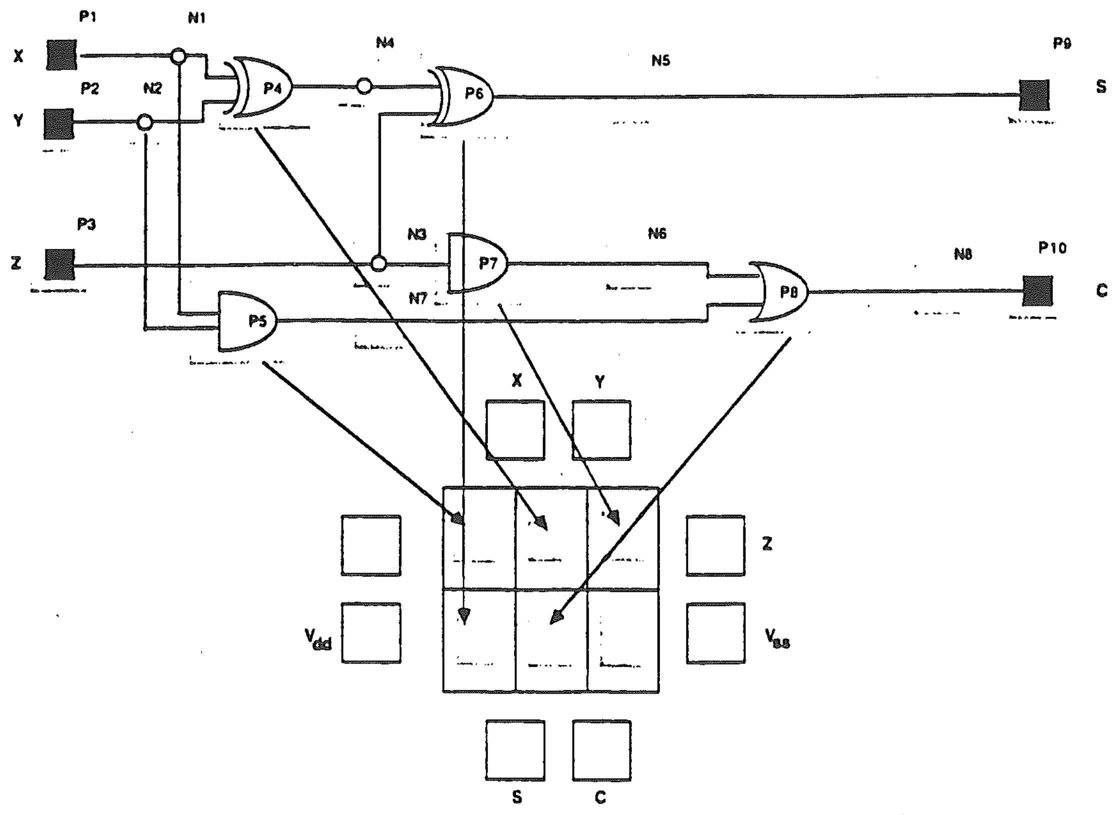

Figure 1: Representation of Circuit and Slots on Surface of Chip on Processing Elements (Shaded areas represent processing elements)

Data Parallel Algorithms for Engineering Applications

Rolf-Dieter Fiebrich
Thinking Machines Corporation
245 First Street
Cambridge, MA 02142

E/G87-1

Abstract

This paper shows how computation-intensive engineering problems can be computed efficiently on a massively parallel computer. Designed with tens of thousands of processing elements, these machines now offer substantially improved computation times and improved cost performance, and in the future promise to rapidly reach higher performance levels. We will discuss the ease of developing data parallel algorithms for VLSI circuit placement, VLSI circuit simulation and seismic migration.

1 Introduction

Massively parallel computers with tens of thousands of processing elements which can operate in parallel are entering the market place. Because of the regularity and simplicity of the design, these machines offer a substantial cost performance improvement over conventional machines. Furthermore, these machines promise to reach substantially higher absolute performance levels in the coming years than conventional supercomputers.

Key questions are often raised: Are computation-intensive applications parallelizable such that these machines can be used efficiently? Can these machines be programmed with reasonable effort?

This paper will address these questions by discussing in some detail three examples of computation intensive algorithms from the engineering domain, namely VLSI circuit placement, VLSI circuit simulation and seismic migration. A goal is to provide some basic insight into the process of designing algorithms for a massively parallel computer from which the reader can extrapolate how other applications can be computed on such a machine.

Virtually all computation-intensive algorithms deal with large amounts of data. VLSI design deals with the placement of thousands of circuit elements or with the simulation of networks of hundreds of thousands of transistors. Seismic migration computes millions of gridpoints of a subsurface image, etc. Massively parallel computing takes advantage of the fact that in those applications, thousands of data objects can be operated on in parallel.

Essential to the ease of parallel programming is a simple model for parallel computation which is efficiently implemented by the underlying hardware. We will use the Connection Machine® system[1] as an example to briefly summarize this model. A programmer starts with familiar concepts for structuring complex data objects such as graphs for representing circuit schematics or arrays for representing images. The new concept is that the variables which represent the vertices of a graph or the elements of an array can be declared to be parallel variables. The other essential concepts are parallel operations which are performed on all parallel variables simultaneously and an operation for selecting a subset of parallel variables in order to restrict parallel operations.

The Connection Machine which supports this model can be viewed as an extension of the memory of a conventional front end, e.g., a VAX, where partitions of the memory have their own execution units which execute parallel operations. In addition, the Connection Machine system has a very sophisticated communications network which permits all parallel execution units to access individual memory locations in the entire machine in parallel.

Parallel variables are stored in individual processing elements, i.e., processor memory pairs. Parallel operations are executed by the parallel execution units. The communication system is involved if compound operations are performed on parallel variables which involve access to other variables; via pointers, for instance.

® Connection Machine is a registered trademark of Thinking Machines Corporation

With this representation of objects in processing elements, the algorithm preceeds as follows:

```
Select randomly a subset (e.g., 40%) of parts
/* selection of new state */
Parts (selected) randomly select a new slot to move towards
/*Parts swap positions with parts found in the selected slot*/
/*the implementation guarantees that parts found in new slots
are disjoint from the originally selected parts*/
/* calculation of change in wirelength (is done individually for
each swapping pair) */
Parts send their new position to nodes
Nodes calculate new wirelength and return results to parts
Parts calculate change in wirelength for all nodes they are con-
nected to
Parts calculate change in wirelength and associated cost for
swapping pairs
/* calculation of rowlength change */
Parts compute change of rowlength and associated cost for each
swapping pair
Parts compute final cost for swaps
/* accept or reject */
Parts compute whether to accept or reject swap
Moves are finalized by updating data in parts and slots
```

With the parallel implementation of the annealing-based algorithm we have tried to stay close to the first two stages of the TimberWolf algorithm[3]. Instead of trying to only permit independent moves, we have implemented an algorithm which tolerates some error in the cost calculation. Errors are due to the interference of parallel swaps (e.g., two swapping pairs may share a common net). Compared to an algorithm with only independent swaps which was outlined for a simple version of an annealing-based algorithm in Reference 4, the above algorithm uses less time per iteration and permits more parallel moves per iteration. During the initial stage of the annealing process approximately 75% of the parts participate in attempted moves. As the temperature decreases the algorithm decreases this percentage as a global means to reduce the error. At the same time the number of iterations per temperature step is increased. Specific parameters for these functions were determined experimentally.

The optimization results of this algorithm are practically the same as for TimberWolf. The execution time of this algorithm for a circuit with more than 6,000 parts is approximately 2 hours on a Connection Machine configuration with 32K processors. On a full 64K processor configuration a circuit with approximately 15,000 parts (40K gate equivalents) can be placed in 2 hours.

3 VLSI Circuit Simulation

Today's VLSI circuits commonly have tens to hundreds of thousands of transistors on a single chip. For accurate verification of the timing behavior, circuits are simulated at the level of analog waveforms. This simulation is so computation-intensive that typically only circuits with 50 to several hundred transistors are simulated at a time.

The circuit simulation problem is formulated as a system of nonlinear differential equations[5]. A sparse matrix whose size depends on the circuit size needs to be solved in the innerloop of the algorithm which solves this problem.

In this paper we discuss a relaxation-based approach[6,7,8,9] to solving this problem, specifically a Gauss-Jacobi relaxation at the nonlinear equation level[10]. This iterative method permits parallel calculation of all unknown voltages of a circuit in the innerloop of the algorithm.

The circuit to be simulated is represented in the machine by storing each device and each node in a separate processing element (Fig. 2). Pointers or references represent the connectivity between devices and nodes.

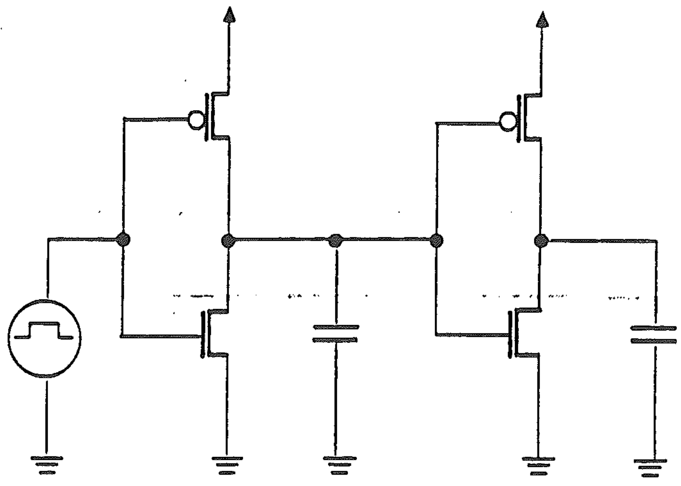

Figure 2: Representation of Circuit on Processing Elements
(Shaded areas represent processing elements)

With this representation of the circuit, the algorithm proceeds as follows:

For each time point the following steps are iterated until convergence is reached

```
all nodes send their voltages to the device terminals (Fig. 3)
all devices compute their new parameters (Fig. 4)
all devices send their parameters to the nodes (Fig. 5)
all nodes compute their new voltages (Fig. 6)
```

Circuits with more than 10,000 devices can be simulated in a few minutes on a Connection Machine system with 64K processors. The performance of the implementation can be improved by storing nodes and parts in the same physical processor since only one type of object is computing at any point in time.

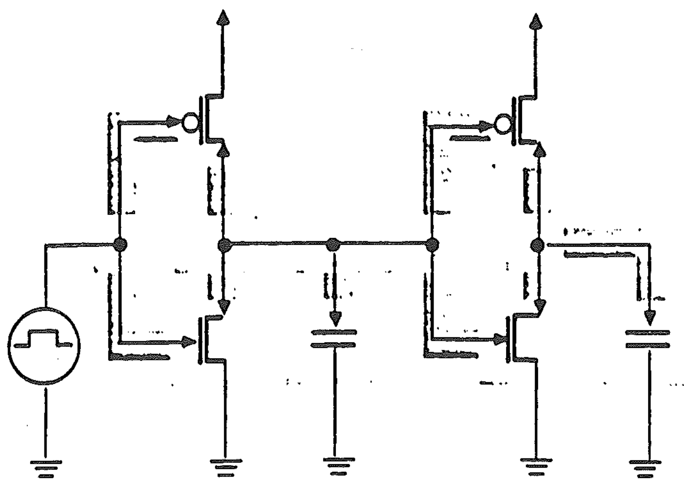
Figure 3: Nodes send Voltages to Devices

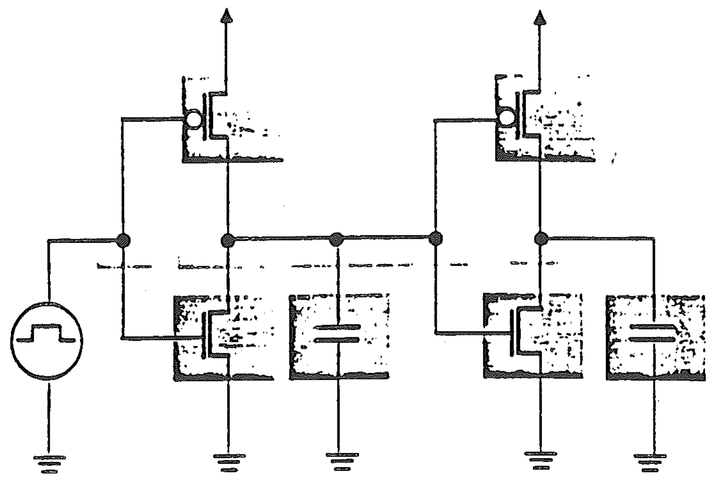
Figure 4: Devices Compute new Parameters

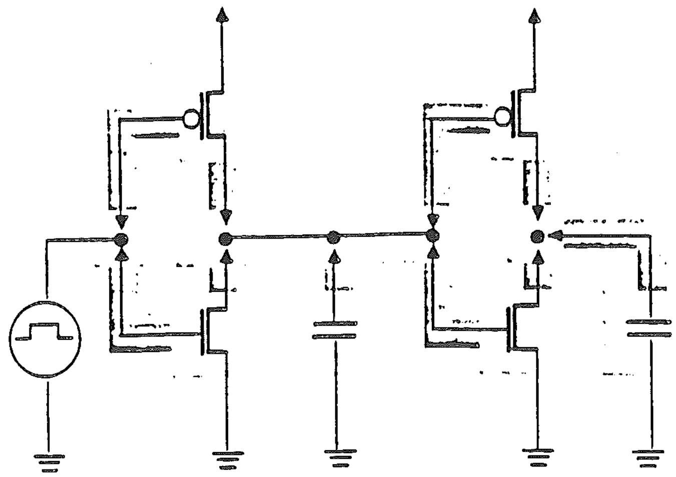
Figure 5: Devices send Parameters to Nodes

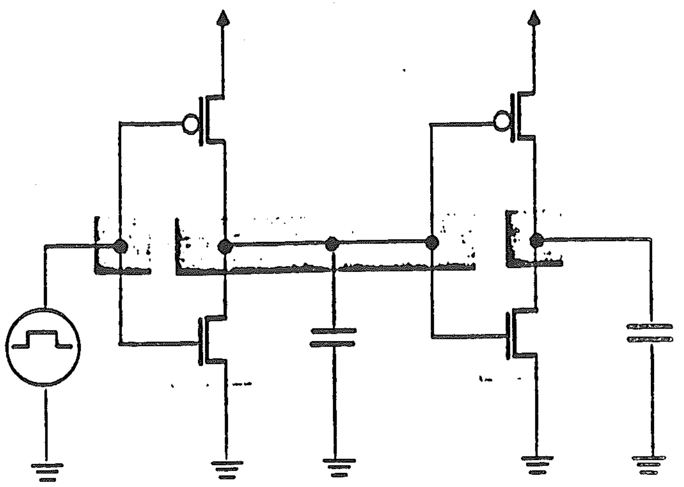
Figure 6: Nodes Compute new Voltages

4 Seismic Migration

For oil and gas exploration geophysicists use seismic methods for acquiring subsurface structural and stratigraphic information without drilling wells. Propagating elastic waves are recorded at the surface of the earth and subsequently processed in order to obtain an image of the subsurface. Seismic migration is a space and time variant filtering process which maps observed space-time amplitude data into depth with correct amplitudes at true spatial positions.[11]

The method for subsurface imaging discussed in this paper is a reverse time migration method[12]; specifically we deal with the acoustic case in which wave propagation is governed by the acoustic wave equation. The algorithm is based on a boundary value solution of the 2-D acoustic wave equation

$$U_{yy} + U_{zz} = \frac{1}{v^2(y,z)} U_{tt}$$

where U is the acoustic wave field. A finite difference scheme is used to discretize the continuous equation. The difference operator chosen is fourth order in both space dimensions and is second order in time. The notation

$$U^k_{i,j} = U(y_i, z_j, t_k)$$

is used in writing the difference operator as follows:

$$\begin{aligned} U^k_{i,j} \;=\;& 2U^{k-1}_{i,j} - U^{k-2}_{i,j} \\ & + \frac{A^2}{12}\,[\,16(U^{k-1}_{i+1,j} + U^{k-1}_{i-1,j} + U^{k-1}_{i,j+1} + U^{k-1}_{i,j-1}) \\ & - 60U^{k-1}_{i,j} \\ & - U^{k-1}_{i+2,j} - U^{k-1}_{i-2,j} - U^{k-1}_{i,j+2} - U^{k-1}_{i,j-2}\,] \end{aligned}$$

where

$$A = v(y,z)\frac{\Delta t}{\Delta y}.$$

In this equation v is the velocity.

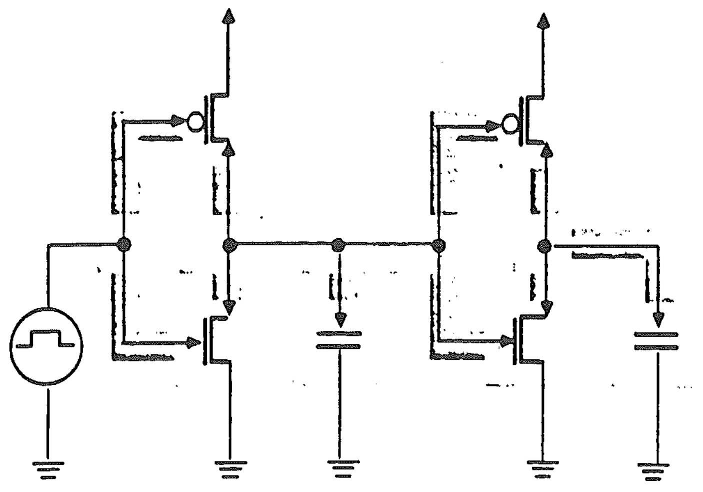
Figure 3: Nodes send Voltages to Devices

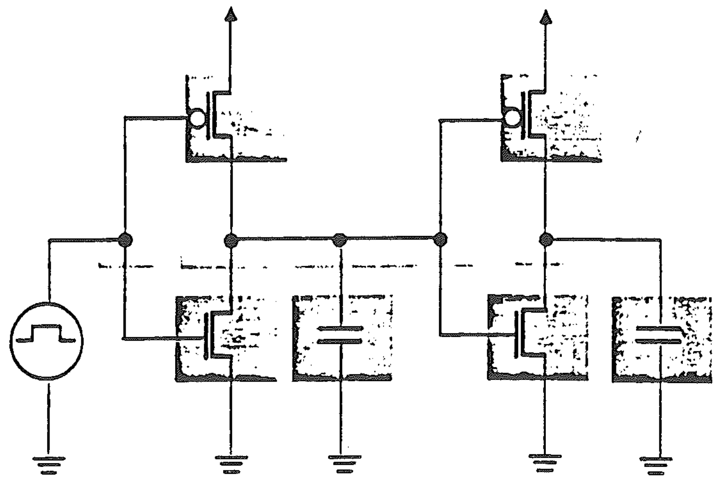
Figure 4: Devices Compute new Parameters

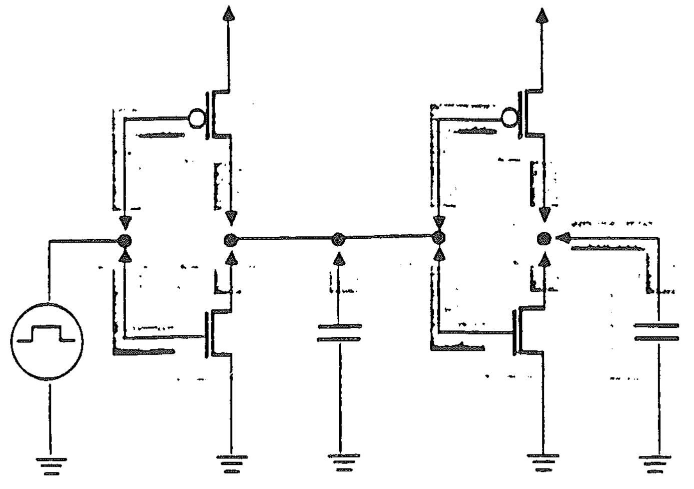
Figure 5: Devices send Parameters to Nodes

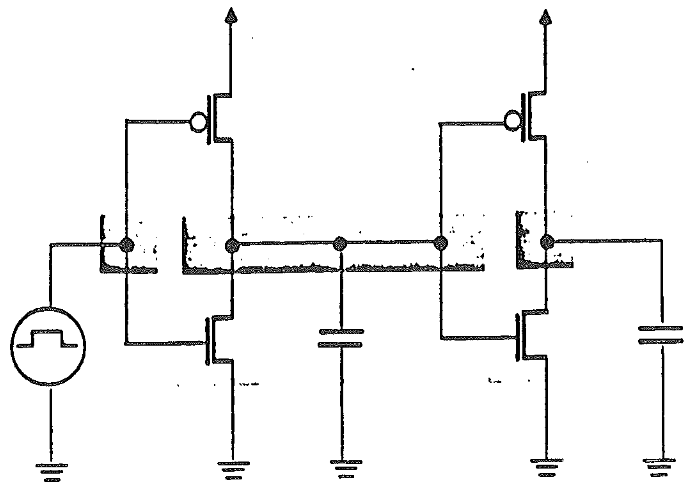
Figure 6: Nodes Compute new Voltages

4 Seismic Migration

For oil and gas exploration geophysicists use seismic methods for acquiring subsurface structural and stratigraphic information without drilling wells. Propagating elastic waves are recorded at the surface of the earth and subsequently processed in order to obtain an image of the subsurface. Seismic migration is a space and time variant filtering process which maps observed space-time amplitude data into depth with correct amplitudes at true spatial positions.[11]

The method for subsurface imaging discussed in this paper is a reverse time migration method[12]; specifically we deal with the acoustic case in which wave propagation is governed by the acoustic wave equation. The algorithm is based on a boundary value solution of the 2-D acoustic wave equation

$$U_{yy} + U_{zz} = \frac{1}{v^2(y,z)} U_{tt}$$

where U is the acoustic wave field. A finite difference scheme is used to discretize the continuous equation. The difference operator chosen is fourth order in both space dimensions and is second order in time. The notation

$$U^k_{i,j} = U(y_i, z_j, t_k)$$

is used in writing the difference operator as follows:

$$\begin{aligned} U^k_{i,j} \; = \; & 2U^{k-1}_{i,j} - U^{k-2}_{i,j} \\ & + \frac{A^2}{12} [\, 16(U^{k-1}_{i+1,j} + U^{k-1}_{i-1,j} + U^{k-1}_{i,j+1} + U^{k-1}_{i,j-1}) \\ & \quad - 60U^{k-1}_{i,j} \\ & \quad - U^{k-1}_{i+2,j} - U^{k-1}_{i-2,j} - U^{k-1}_{i,j+2} - U^{k-1}_{i,j-2} \,] \end{aligned}$$

where

$$A = v(y,z)\frac{\Delta t}{\Delta y}.$$

In this equation v is the velocity.

With this representation of objects in processing elements, the algorithm preceeds as follows:

```
Select randomly a subset (e.g., 40%) of parts
/* selection of new state */
Parts (selected) randomly select a new slot to move towards
/*Parts swap positions with parts found in the selected slot*/
/*the implementation guarantees that parts found in new slots
are disjoint from the originally selected parts*/
/* calculation of change in wirelength (is done individually for
each swapping pair) */
Parts send their new position to nodes
Nodes calculate new wirelength and return results to parts
Parts calculate change in wirelength for all nodes they are con-
nected to
Parts calculate change in wirelength and associated cost for
swapping pairs
/* calculation of rowlength change */
Parts compute change of rowlength and associated cost for each
swapping pair
Parts compute final cost for swaps
/* accept or reject */
Parts compute whether to accept or reject swap
Moves are finalized by updating data in parts and slots
```

With the parallel implementation of the annealing-based algorithm we have tried to stay close to the first two stages of the TimberWolf algorithm[3]. Instead of trying to only permit independent moves, we have implemented an algorithm which tolerates some error in the cost calculation. Errors are due to the interference of parallel swaps (e.g., two swapping pairs may share a common net). Compared to an algorithm with only independent swaps which was outlined for a simple version of an annealing-based algorithm in Reference 4, the above algorithm uses less time per iteration and permits more parallel moves per iteration. During the initial stage of the annealing process approximately 75% of the parts participate in attempted moves. As the temperature decreases the algorithm decreases this percentage as a global means to reduce the error. At the same time the number of iterations per temperature step is increased. Specific parameters for these functions were determined experimentally.

The optimization results of this algorithm are practically the same as for TimberWolf. The execution time of this algorithm for a circuit with more than 6,000 parts is approximately 2 hours on a Connection Machine configuration with 32K processors. On a full 64K processor configuration a circuit with approximately 15,000 parts (40K gate equivalents) can be placed in 2 hours.

3 VLSI Circuit Simulation

Today's VLSI circuits commonly have tens to hundreds of thousands of transistors on a single chip. For accurate verification of the timing behavior, circuits are simulated at the level of analog waveforms. This simulation is so computation-intensive that typically only circuits with 50 to several hundred transistors are simulated at a time.

The circuit simulation problem is formulated as a system of nonlinear differential equations[6]. A sparse matrix whose size depends on the circuit size needs to be solved in the innerloop of the algorithm which solves this problem.

In this paper we discuss a relaxation-based approach[6,7,8,9] to solving this problem, specifically a Gauss-Jacobi relaxation at the nonlinear equation level[10]. This iterative method permits parallel calculation of all unknown voltages of a circuit in the innerloop of the algorithm.

The circuit to be simulated is represented in the machine by storing each device and each node in a separate processing element (Fig. 2). Pointers or references represent the connectivity between devices and nodes.

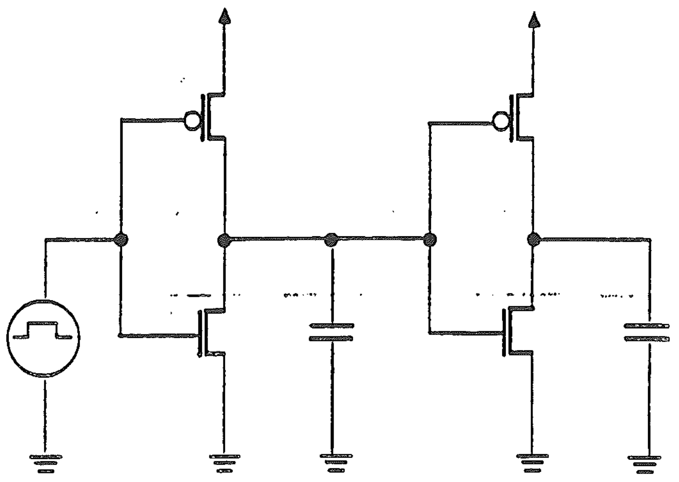

Figure 2: Representation of Circuit on Processing Elements (Shaded areas represent processing elements)

With this representation of the circuit, the algorithm proceeds as follows:

For each time point the following steps are iterated until convergence is reached

```
all nodes send their voltages to the device terminals (Fig. 3)
all devices compute their new parameters (Fig. 4)
all devices send their parameters to the nodes (Fig. 5)
all nodes compute their new voltages (Fig. 6)
```

Circuits with more than 10,000 devices can be simulated in a few minutes on a Connection Machine system with 64K processors. The performance of the implementation can be improved by storing nodes and parts in the same physical processor since only one type of object is computing at any point in time.

Figure 7 illustrates the computation for this method. For each time step the depth section needs to be computed. This computation involves references to the depth section of the two previous time steps, the row of the source data for the corresponding time step and the velocity model. Figure 8 shows the computation of individual grid points of the depth section. The Figure shows a simplified computation of a second order finite difference scheme, whereas the actual implementation is fourth order. As one can see, all grid points of the depth section for a given time point can be computed in parallel. The following summarizes a data parallel algorithm. A more detailed description can be found in Reference 13.

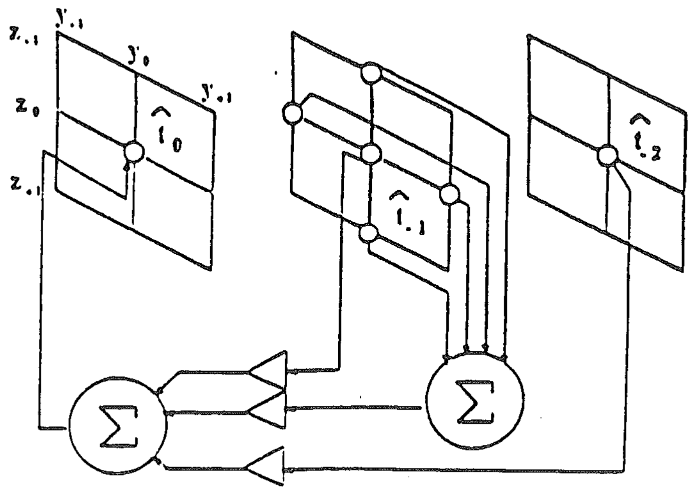

Figure 8: Computation of Second Order Difference Operator

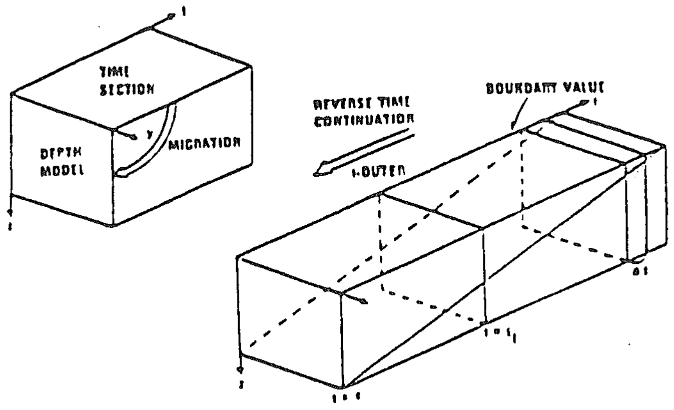

Figure 7: Reverse Time Migration

A processing element is assigned to each grid point in the depth section. The processing element holds also the data for this grid point which corresponds to the depth sections of the previous time points and of the velocity model. The source data is stored by distributing it over the processing elements of the depth section. Fig. 9 summarizes the mapping of data objects to processing elements in this problem.

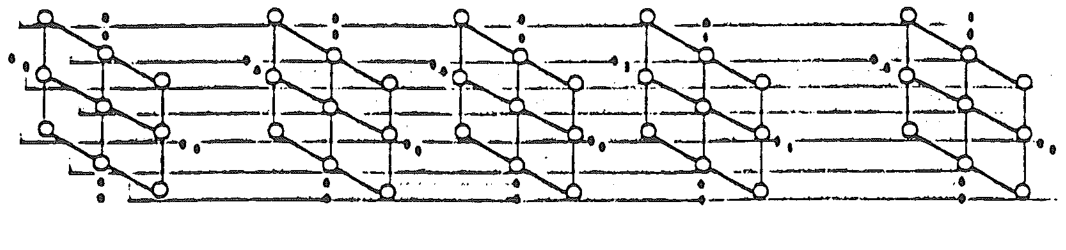

Figure 9: Representation of Depth Section, Velocity Model and Source Data on Processing Elements

(Shaded areas represent processing elements)

The following computation takes place for each time step. The source data array is shifted up one step. The row which is shifted out at the top is placed onto the top of the depth section. All processing elements compute a depth point in parallel accessing their nearest and second nearest neighbors in two space dimensions and in the time dimension.

The actual implementation of this algorithm utilizes more than 75% of the peak performance of the Connection Machine.

5 Conclusion

We have discussed the development of algorithms for a massively parallel computer. Problems such as circuit placement, circuit simulation and seismic migration have significantly more inherent parallelism than conventional computers or even vector machines can utilize. Hence, a massively parallel machine can provide substantial speedups for those problems. This speedup also makes possible applications which have been prohibitive so far. Much larger problems can be solved, for instance the simulation of complete chips at the analog level.

The data parallel programming style is effectively supported by massively parallel machines and makes program development easy. Associating a data object with a processing element makes it straightforward to decompose a task into subtasks which can be executed in parallel. A key step in developing data parallel algorithms is to define appropriate data structures and an appropriate mapping of those structures onto processing elements. Since the data objects which are computed in parallel are generally of the same type, a single program control stream is sufficient, i.e., no synchronization problems between concurrent processes need to be solved by the programmer.

6 Acknowledgements

I am grateful for discussions on data parallel algorithms with Alberto Sangiovanni-Vincentelli, Don Webber, Andrea Casotto, Chi Pong Wong, Monica Wong, Rob Fricke, and Wok Lee. I thank Olivia Fiske and Jim Bailey for their assistance in editing this paper.

7 References

1. W. D. Hillis, *The Connection Machine*, The MIT Press, Cambridge, MA, 1985
2. S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, May 1983.
3. C. Sechen and A. L. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *IEEE Journal of Solid-State Circuits*, Vol.SC-20, No. 2, April 1985
4. R. D. Fiebrich, "A Supercomputer Workstation for VLSI CAD," *IEEE Design and Test*, June 1986
5. L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," Memorandum No. ERL-M520, College of Engineering, University of California, Berkeley, May 1975
6. R. A. Newton and A. L. Sangiovanni-Vincentelli, "Relaxation-Based Electrical Simulation," *IEEE Trans. Computer-Aided Design*, Vol. CAD-3, No. 4, Oct. 1984
7. A. E. Ruehli and G. S. Ditlow, "Circuit Analysis, Logic Simulation, and Design Verification for VLSI," *IEEE Proceedings*, Vol. 71, No. 1, Jan. 1983
8. E. Lelarasmee, "The Waveform Relaxation Method for Time Domain Analysis of Large Scale Integrated Circuits: Theory and Applications," Memorandum No. UCB/ERL M82/40, College of Engineering, University of California, Berekeley, May 1982
9. J. White, "The Multirate Integration Properties of Waveform Relaxation with Applications to Circuit Simulation and Parallel Computation," Ph.D. Dissertation, Electronics Research Laboratory, University of California, Berekeley, November 1985
10. D. Webber, "An Introduction to Circuit Simulation with Algorithms for the Connection Machine," Thinking Machines Technical Memo, 85.7.11:W, Thinking Machines Corporation, 245 First Street, Cambridge, MA 1985
11. *Concepts and Techniques in Oil and Gas Exploration*, K. C. Jain and R. J. P. deFigueiredo, (eds.), Society of Exploration Geophysicists, 1982
12. G. A. McMechan, "Migration by Extrapolation of Time-Dependent Boundary Values," Geophysical Prospecting 31, 1983
13. J. Robert Fricke, "Reverse Time Migration in Parallel," Thinking Machines Corporation Technical Report, Oct 1986

The following computation takes place for each time step. The source data array is shifted up one step. The row which is shifted out at the top is placed onto the top of the depth section. All processing elements compute a depth point in parallel accessing their nearest and second nearest neighbors in two space dimensions and in the time dimension.

The actual implementation of this algorithm utilizes more than 75% of the peak performance of the Connection Machine.

5 Conclusion

We have discussed the development of algorithms for a massively parallel computer. Problems such as circuit placement, circuit simulation and seismic migration have significantly more inherent parallelism than conventional computers or even vector machines can utilize. Hence, a massively parallel machine can provide substantial speedups for those problems. This speedup also makes possible applications which have been prohibitive so far. Much larger problems can be solved, for instance the simulation of complete chips at the analog level.

The data parallel programming style is effectively supported by massively parallel machines and makes program development easy. Associating a data object with a processing element makes it straightforward to decompose a task into subtasks which can be executed in parallel. A key step in developing data parallel algorithms is to define appropriate data structures and an appropriate mapping of those structures onto processing elements. Since the data objects which are computed in parallel are generally of the same type, a single program control stream is sufficient, i.e., no synchronization problems between concurrent processes need to be solved by the programmer.

6 Acknowledgements

I am grateful for discussions on data parallel algorithms with Alberto Sangiovanni-Vincentelli, Don Webber, Andrea Casotto, Chi Pong Wong, Monica Wong, Rob Fricke, and Wok Lee. I thank Olivia Fiske and Jim Bailey for their assistance in editing this paper.

7 References

1. W. D. Hillis, *The Connection Machine*, The MIT Press, Cambridge, MA, 1985
2. S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, May 1983.
3. C. Sechen and A. L. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *IEEE Journal of Solid-State Circuits*, Vol.SC-20, No. 2, April 1985
4. R. D. Fiebrich, "A Supercomputer Workstation for VLSI CAD," *IEEE Dcsign and Test*, June 1986
5. L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," Memorandum No. ERL-M520, College of Engineering, University of California, Berkeley, May 1975
6. R. A. Newton and A. L. Sangiovanni-Vincentelli, "Relaxation-Based Electrical Simulation," *IEEE Trans. Computer-Aided Design*, Vol. CAD-3, No. 4, Oct. 1984
7. A. E. Ruehli and G. S. Ditlow, "Circuit Analysis, Logic Simulation, and Design Verification for VLSI," *IEEE Proceedings*, Vol. 71, No. 1, Jan. 1983
8. E. Lelarasmee, "The Waveform Relaxation Method for Time Domain Analysis of Large Scale Integrated Circuits: Theory and Applications," Memorandum No. UCB/ERL M82/40, College of Engineering, University of California, Berekeley, May 1982
9. J. White, "The Multirate Integration Properties of Waveform Relaxation with Applications to Circuit Simulation and Parallel Computation," Ph.D. Dissertation, Electronics Research Laboratory, University of California, Berekeley, November 1985
10. D. Webber, "An Introduction to Circuit Simulation with Algorithms for the Connection Machine," Thinking Machines Technical Memo, 85.7.11:W, Thinking Machines Corporation, 245 First Street, Cambridge, MA 1985
11. *Concepts and Techniques in Oil and Gas Exploration*, K. C. Jain and R. J. P. deFigueiredo, (eds.), Society of Exploration Geophysicists, 1982
12. G. A. McMechan, "Migration by Extrapolation of Time-Dependent Boundary Values," Geophysical Prospecting 31, 1983
13. J. Robert Fricke, "Reverse Time Migration in Parallel," Thinking Machines Corporation Technical Report, Oct 1986

Figure 7 illustrates the computation for this method. For each time step the depth section needs to be computed. This computation involves references to the depth section of the two previous time steps, the row of the source data for the corresponding time step and the velocity model. Figure 8 shows the computation of individual grid points of the depth section. The Figure shows a simplified computation of a second order finite difference scheme, whereas the actual implementation is fourth order. As one can see, all grid points of the depth section for a given time point can be computed in parallel. The following summarizes a data parallel algorithm. A more detailed description can be found in Reference 13.

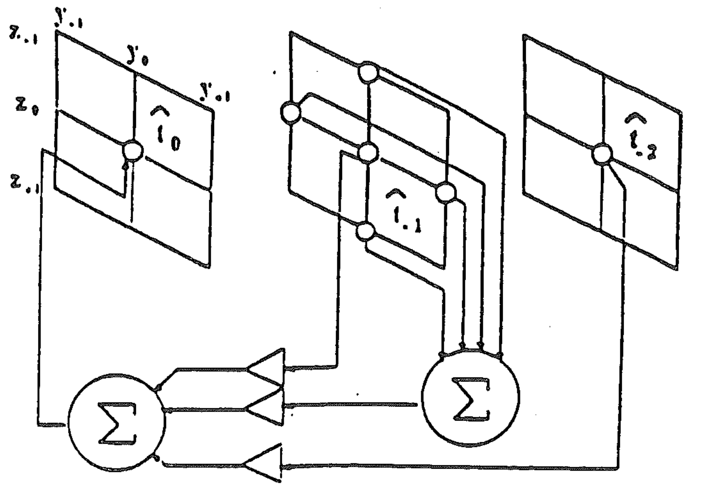

Figure 8: Computation of Second Order Difference Operator

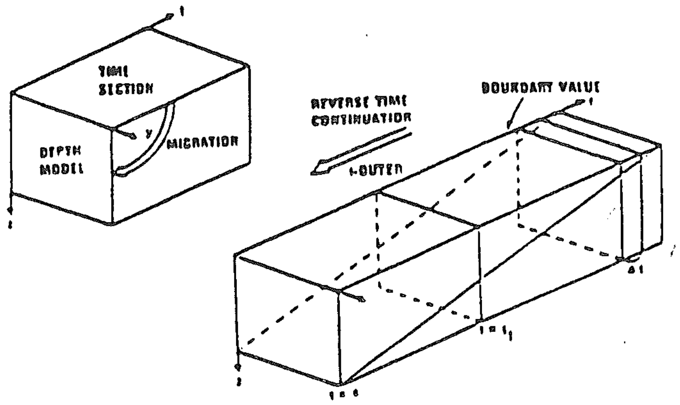

Figure 7: Reverse Time Migration

A processing element is assigned to each grid point in the depth section. The processing element holds also the data for this grid point which corresponds to the depth sections of the previous time points and of the velocity model. The source data is stored by distributing it over the processing elements of the depth section. Fig. 9 summarizes the mapping of data objects to processing elements in this problem.

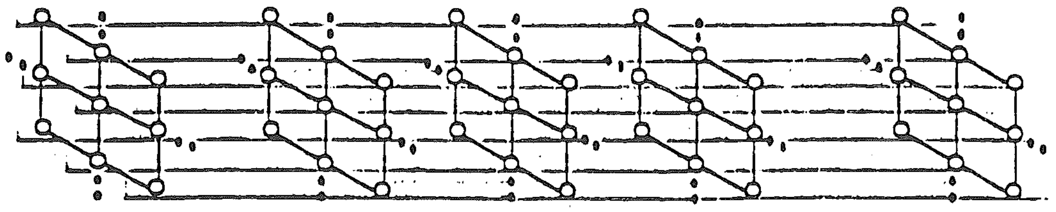

Figure 9: Representation of Depth Section, Velocity Model and Source Data on Processing Elements
(Shaded areas represent processing elements)